

# MULTIPROD TOOLBOX

## [MATLAB® evolves becoming ARRAYLAB ☺]

### Multiple matrix multiplications, with array expansion enabled

*Paolo de Leva*

University of Rome – Foro Italico, Rome, IT

#### Summary

Multiple products between **matrices**, **vectors**, or **scalars** contained in two block arrays (fig. 1), with automatic virtual array expansion enabled.

#### Description

MULTIPROD is a powerful, quick and memory efficient generalization for N-D arrays of the MATLAB matrix multiplication operator (\*). While the latter operates only on 2-D arrays, MULTIPROD also operates on multi-dimensional arrays.

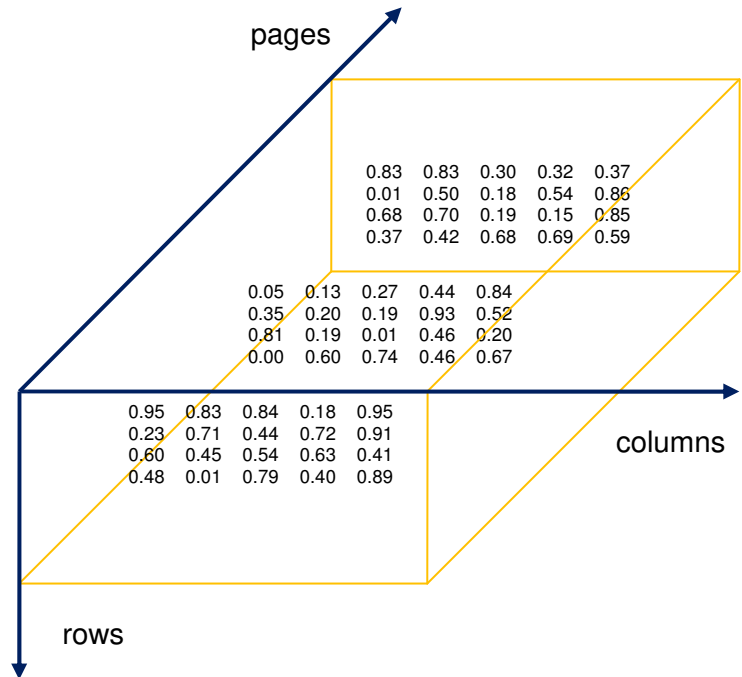
Besides the element-wise multiplication operator (. \*), MATLAB includes only two functions which can perform products between multidimensional arrays: DOT and CROSS. However, these functions can only perform two kinds of product: the dot product and the cross product, respectively. Also, they cannot apply array expansion.

Conversely, MULTIPROD can perform any kind of multiple scalar-by-matrix or matrix multiplication:

- 1) Arrays of **scalars** by arrays of **scalars**, **vectors** (\*) or **matrices**.
  - 2) Arrays of **vectors** (\*) by arrays of **scalars**, **vectors** (\*) or **matrices**.
  - 3) Arrays of **matrices** by arrays of **scalars**, **vectors** (\*) or **matrices**.
- (\*) internally converted by MULTIPROD into row or column matrices.

In short, with MULTIPROD the "matrix laboratory" **MATLAB®** evolves becoming **ARRAYLAB ☺**, an "array laboratory". Moreover, MULTIPROD is capable of automatically applying virtual "array expansion" (AX), which allows you, for instance, to multiply a single matrix **A** by an array of matrices **B**, by virtually replicating the matrix to obtain an array compatible with **B**.

Multidimensional arrays may contain **matrices** or **vectors** or even **scalars** along one or two of their dimensions. For instance, a  $4 \times 5 \times 3$  array **A** contains three  $4 \times 5$  matrices along its first and second dimension (fig. 1). Thus, array **A** can be described as a *block array* the elements of which are matrices, and its size can be denoted by  $(4 \times 5) \times 3$ .



**Figure 1.** A 3-D array, with size  $4 \times 5 \times 3$ , may be described as a "block array" containing three  $4 \times 5$  matrices (one per page), or also four  $5 \times 3$  matrices (one per row). With MULTIPROD, these matrices can be automatically multiplied by matrices, vectors or scalars contained in another array.

MULTIPROD can be also described as a generalization of the built-in function `TIMES`. While `TIMES` operates element-by-element multiplications (e.g.  $\mathbf{A} \cdot \mathbf{B}$ ), `MULTIPROD` operates block-by-block matrix multiplications.

## Examples

Let's say that

- ✚  $\mathbf{A}$  is  $(2 \times 5) \times 6$ , and
- ✚  $\mathbf{B}$  is  $(5 \times 3) \times 6$ .

With `MULTIPROD` the six **matrices** in  $\mathbf{A}$  can be multiplied by those in  $\mathbf{B}$  in a single intuitively appealing step:

$$\mathbf{C} = \text{MULTIPROD}(\mathbf{A}, \mathbf{B}).$$

where  $\mathbf{C}$  is  $(2 \times 3) \times 6$ .

By automatically applying AX, `MULTIPROD` can multiply a single matrix by each of the blocks of a block array. So, if

- ✚  $\mathbf{A}$  is  $2 \times 5$  (single matrix), and
- ✚  $\mathbf{B}$  is  $(5 \times 3) \times 1000 \times 10$ ,

then  $\mathbf{C} = \text{MULTIPROD}(\mathbf{A}, \mathbf{B})$  yields a  $(2 \times 3) \times 1000 \times 10$  array.  $\mathbf{A}$  is virtually expanded to a  $(2 \times 5) \times 1000 \times 10$  size, then multi-multiplied by  $\mathbf{B}$ . This is done without using loops, and without actually replicating the matrix (see Appendix A). We refer to this particular application of AX as virtual matrix expansion. In a system running MATLAB R2008a, `MULTIPROD` performs it about 380 times faster than the simple loop shown in Fig. 2 (see Appendix B). AX generalizes matrix expansion to multidimensional arrays of any size. For instance, if

- ✚  $\mathbf{A}$  is  $(2 \times 5) \times 10$ , and
- ✚  $\mathbf{B}$  is  $(5 \times 3) \times 1 \times 6$ ,

then  $\mathbf{C} = \text{MULTIPROD}(\mathbf{A}, \mathbf{B})$  multiplies *each* of the 10 matrices in  $\mathbf{A}$  by *each* of the 6 matrices in  $\mathbf{B}$ , obtaining 60 matrices stored in a  $(2 \times 3) \times 10 \times 6$  array  $\mathbf{C}$ . It does that by virtually expanding  $\mathbf{A}$  to  $(2 \times 5) \times 10 \times 6$ , and  $\mathbf{B}$  to  $(5 \times 3) \times 10 \times 6$ . A detailed definition of AX will be given in a separate section.

Here are a few other examples of block arrays on which `MULTIPROD` can operate, including arrays with 1-D blocks (D and E), and scalar blocks (C and E):

- ✚  $\mathbf{A}$  is a  $2 \times 5 \times (6 \times 3)$  array containing **matrices** along dimensions 3 and 4.
- ✚  $\mathbf{B}$  is a  $2 \times 5 \times (3 \times 4)$  array containing **matrices** along dimensions 3 and 4.
- ✚  $\mathbf{C}$  is a  $2 \times 5 \times (1 \times 1)$  array containing **scalars** along dimensions 3 and 4.
- ✚  $\mathbf{D}$  is a  $2 \times 5 \times (3)$  array containing **vectors** along dimension 3.
- ✚  $\mathbf{E}$  is a  $2 \times 5 \times (1)$  array containing **scalars** along dimension 3.

For instance, `MULTIPROD` can multiply the 10 **matrices** in  $\mathbf{A}$  by the 10 **matrices**, **vectors** or **scalars** occupying the same position in  $\mathbf{B}$ ,  $\mathbf{C}$ ,  $\mathbf{D}$ , or  $\mathbf{E}$  (in this case, the vectors in  $\mathbf{D}$  are regarded as  $3 \times 1$  matrices).

The help text of `MULTIPROD` was written with extreme care, and should be enough for those who just want to use the function. Details on the algorithm are provided in Appendices A, B, and C.

```
% Building A and B
A = rand(2, 5);
B = rand(5, 3, 1000, 10);

% Multiplying A by all the matrices in B
for i = 1:1000
    for j = 1:10
        C(:, :, i, j) = A * B(:, :, i, j);
    end
end
```

**Figure 2.** This loop is equivalent to the single instruction  $\mathbf{C} = \text{MULTIPROD}(\mathbf{A}, \mathbf{B})$ , but `MULTIPROD` performs the same task about 380 times faster. You can test this example on your system by running function **timing\_MX**, included in this toolbox.

## Internal and external dimensions

In this context, a block array is defined as an array the elements of which are **matrices**, **vectors** or **scalars** (fig. 1). Each of these elements is referred to as a block.

The one or two adjacent dimensions of the block array along which the blocks are contained are called *internal dimensions* (IDs), while all its other dimensions are called *external dimensions* (EDs).

For instance, if **A** is a  $(4 \times 5) \times 3$  array of **matrices**, its first two dimensions are internal (IDs = [1 2]), and the third is external (ED = 3).

## Array expansion

AX is a generalization to N-D of the concept of scalar expansion. Indeed, **A** and **B** may be scalars, vectors, matrices or multi-dimensional arrays.

In MATLAB, scalar expansion is the virtual replication or annihilation of a scalar which allows you to combine it to an array **X** of any size, including empty arrays. For instance, if **X** is  $2 \times 5 \times 6$ , the operation  $\mathbf{X} * 10$  multiplies each element of **X** by the scalar 10, which is virtually equivalent to an *element-by-element* multiplication  $\mathbf{X} .* \mathbf{Y}$ , where **Y** is a  $2 \times 5 \times 6$  matrix filled with tens. On the other hand, if **X** is empty,  $\mathbf{X} * 10 = \mathbf{X}$ , which is virtually equivalent to annihilating the scalar and multiplying **X** element-wise by another empty array **Y** of the same size.

Similarly, in MULTIPROD, the purpose of AX is to virtually match the size of the external dimensions (EDs) of **A** and **B**, so that *block-by-block* products can be performed. ED matching is achieved by means of a dimension shift followed by a singleton expansion:

### 1) Dimension shift (see SHIFTDIM).

Whenever  $ID_{A1} \neq ID_{B1}$ , a shift is applied to impose  $ID_{A1} == ID_{B1}$  ( $ID_{A1}$  is the first ID of **A**, and  $ID_{B1}$  is the first ID of **B**; e.g., if the size of **A** is  $6 \times (2 \times 5)$ , then its IDs are [2 3] and  $ID_{A1}$  is 2).





If  $ID_{A1} > ID_{B1}$ , **B** is shifted to the right by  $ID_{A1} - ID_{B1}$  steps.

If  $ID_{B1} > ID_{A1}$ , **A** is shifted to the right by  $ID_{B1} - ID_{A1}$  steps.

### 2) Singleton expansion (SX).

Whenever an ED of either **A** or **B** is singleton and the corresponding ED of the other array is not, the mismatch is fixed by virtually replicating the array (or diminishing it to length 0) along that dimension.

For instance,

	if <b>A</b> is .....	$6 \times (2 \times 5)$ ,
	and <b>B</b> is .....	$(5 \times 3) \times 5$ ,
	then <b>B</b> is shifted by 1 dim. and becomes ....	$1 \times (5 \times 3) \times 5$ ,
	and $\mathbf{C} = \text{MULTIPROD}(\mathbf{A}, \mathbf{B}, ID_A, ID_B)$ is	$6 \times (2 \times 3) \times 5$ .

where  $ID_A = [2\ 3]$  and  $ID_B = [1\ 2]$  are the internal dimensions of **A** and **B**, respectively.

## Applications

MULTIPROD has a broad field of potential applications. By calling MULTIPROD, multiple geometrical transformations such as rotations or roto-translations can be performed on large arrays of vectors in a single step and with no loops. Multiple operations such as normalizing an array of vectors, or finding their projection along the axes indicated by another array of vectors can be performed easily, with no loops and with two or three rows of code.

Sample functions performing some of these tasks by calling MULTIPROD are included in the separate toolbox "Vector algebra for multidimensional arrays of vectors" (MATLAB Central, file #8782). A sample function (LOC2LOC) performing a multiple roto-translation by calling MULTIPROD is included in this package. This function uses 3-element translation vectors and 3x3 rotation matrices. If you prefer to work with homogeneous coordinates and 4x4 roto-translation matrices, you can obtain the same result just by calling MULTIPROD twice.

## Optimization and testing

Since I wanted to be of service to as many people as possible, MULTIPROD was designed, debugged, and optimized for speed and memory efficiency with extreme care. Precious advices by Jinhui Bai (Georgetown University) helped me to make it even faster, more efficient and more versatile. Suggestions to improve it further will be welcome. The code ("testMULTIPROD.m") I used to systematically test the function output is included in this package.

## The ARRAYLAB toolbox

In sum, MULTIPROD is a generalization for N-D arrays of the matrix multiplication function MTIMES, with AX enabled. Vector inner, outer, and cross products generalized for N-D arrays and with AX enabled are performed by DOT2, OUTER, and CROSS2 (MATLAB Central, file #8782). Element-by-element multiplications (see TIMES) and other element-by-element binary operations (such as PLUS and EQ) with AX enabled are performed by BAXFUN (MATLAB Central, file #23084).

Together, these functions make up the "ARRAYLAB toolbox". I hope that The MathWorks will include it in the next version of MATLAB.

## Requirements

The users of MATLAB releases prior to R2007a must install Douglas Schwarz's substitute for **bsxfun**, a function which is builtin in MATLAB R2007a and later releases, and which represents the core of one of the two main engines of MULTIPROD.

## MULTITRANSF

This package includes the function MULTITRANSF, performing multiple matrix transpositions.

**B** = MULTITRANSF(**A**, DIM)

transposes all the matrices contained along dimensions DIM and DIM+1 of **A**.

## Acknowledgements

I wish to express my gratitude to Jinhui Bai (Georgetown University, Washington, D.C.) for his invaluable suggestions to optimize the engines used by MULTIPROD, for writing some of the engines tested by the m-function **timing\_arraylab\_engines**, and for patiently running the infinitely many versions of that function on his laptop computer.

On behalf of all the users of pre-R2007a MATLAB releases (including me), I also wholeheartedly thank Douglas Schwarz for making the latest version of MULTIPROD fully compatible with those releases. He did so by generously granting my request to publish on MATLAB Central (file #23005) a memory efficient replacement for **bsxfun**, a builtin function introduced in MATLAB R2007a which I exploited to assemble one of the two main engines of MULTIPROD.